Production Scheduling by Reachability Analysis – A Case Study*

Gerd Behrmann[1] Ed Brinksma[2] Martijn Hendriks[3] Angelika Mader[2]

[1] Aalborg University, Aalborg, Denmark
behrmann@cs.auc.dk

[2] University of Twente, Enschede, The Netherlands
H.Brinksma@ewi.utwente.nl, mader@cs.utwente.nl

[3] Radboud University, Nijmegen, The Netherlands
M.Hendriks@cs.ru.nl

Abstract

Schedule synthesis based on reachability analysis of timed automata has received attention in the last few years. The main strength of this approach is that the expressiveness of timed automata allows – unlike many classical approaches – the modelling of scheduling problems of very different kinds. Furthermore, the models are robust against changes in the parameter setting and against changes in the problem specification. This paper presents a case study that was provided by Axxom, an industrial partner of the Ametist project. It consists of a scheduling problem for lacquer production, and is treated with the timed automata approach. A number of problems have to be addressed for the modelling task: the information transfer from the industrial partner, the derivation of timed automaton model for the case study, and the heuristics that have to be added in order to reduce the search space. We try to isolate the generic problems of modelling for model checking, and suggest solutions that are also applicable for other scheduling cases. Model checking experiments indicate that – for this problem – the timed automata approach is competitive with Axxom's planning tool.
Keywords: Scheduling, model checking, timed automata, cost optimization, industrial case study.

1 Introduction

Scheduling theory is a well-established branch of operations research, and has produced a wealth of theory and techniques that can be used to solve many practical problems, such as real-time problems in operating systems, distributed systems, process control, etc. [12, 11]. Despite this success, alternative and complementary approaches to schedule synthesis based on reachability analysis of timed automata have been proposed in the last few years [6, 2, 1]. The main motivation of this

*Supported by the European Community Project IST-2001-35304 Ametist (Advanced Methods for Timed Systems), http://ametist.cs.utwente.nl/.

previous work is the observation that many scheduling problems can very naturally be modelled with timed automata. Furthermore, the expressiveness of timed automata renders the models robust against changes in parameter settings and changes in the problem specification. It has been shown that this approach is not necessarily inferior to other methods developed during the last three decades [2].

The case study presented in this paper is one of the four industrial case studies of the European IST project AMETIST, which focuses on the application of advanced formal methods for the modelling and analysis of complex distributed real-time systems with dynamic resource allocation as one of its special topics. The application of timed reachability analysis to this problem is one of the main subjects of the project. Technical material related to this case study, and different approaches to its solution can be retrieved from the AMETIST website [4].

The remainder of this paper is organized as follows. The principles of the derivation of schedules by reachability analysis are sketched in Section 2. Section 3 contains a description of the case study. Modelling issues and the use of heuristics are discussed in Section 4. The results of our model-checking experiments are collected and discussed in Sections 5 and 6. Finally, Section 7 evaluates the model checking approach to the case study and concludes the paper.

2 Scheduling With Timed Automata

The synthesis of schedules using timed automata can be seen as a special case of control synthesis [10], and was first introduced by [6], and by [2]. In general, a model class that provides the possibility to represent system events as well as timing information is suitable for real-time control synthesis. In this paper, the timed automata of Alur and Dill are used for modelling [3]. These timed automata extend the traditional model of finite automata with real-valued clock variables whose values increase with the rate of the progress of time. Clock variables can be reset to zero and they can be used in guards for discrete transitions as well as in guards for the elapse of time (this is used to ensure progress). In general, timed automata models have an infinite state space. The region automaton construction, however, shows that this infinite state space can be mapped to an automaton with a finite number of equivalence classes (regions) as states [3]. Finite-state model checking techniques can then be applied to the reduced, finite region automaton. A number of model checkers for timed automata is available, for instance, KRONOS [13] and UPPAAL [8].

Schedule synthesis using timed automata works as follows. First, a model of the unscheduled system is constructed. In our case, this model consists of the parallel composition of a number of timed automata. The non-determinism that is present in the parallel composition reflects the unresolved scheduling choices. Second, feasibility is formulated as a reachability property, for instance, "It is possible that the production is finished by Friday evening". Third, the model checker exhaustively searches the reachable state space in order to check whether the property holds. If this is the case, then the model checker can provide a trace that proves the property. In our example, this is a trace from the initial state to

a state in which the production is finished and it is not later than Friday evening. The information contained in such a trace suffices to extract a feasible schedule, which is the final step of our approach.

The advantage of this approach is its (modelling) robustness against changes in the parameter settings and changes in the problem specification. The disadvantage lies in the well-known state space explosion problem: the reachable state space is far too large to handle within a practical amount of time for many interesting cases. The approach that is used in this paper is to add heuristics and to use features of schedules that reduce the reachable state space to a size that can be searched more easily. We argue that these heuristics are quite general and applicable in many cases.

3 Description of the Case Study

The case study consists of a problem that is almost a job-shop problem [11]. The problem is to schedule a number of lacquer orders on a set of resources (machines). Each order is specified by an earliest starting time, a due date and a lacquer type. There are three lacquer types, and each type is produced following a different recipe. A recipe specifies the production steps that are needed to produce the lacquer, and includes all constraints such as resource and time usage and conditions relating start and/or end times of subsequent production steps.

Figure 1 shows the recipe for uni-type lacquers in a graphical representation as given by Axxom. The circles represent processing steps and are annotated with the name of the step, the resource that is needed for that step and the time the step takes. The arrows in Figure 1 have different interpretations, e.g., an arrow labeled with 13 represents the normal precedence constraint while an arrow from A to B labeled with 15 means that A and B have to start at the same time. An alternative notation for the three recipes is introduced in the next section.

As mentioned above, the problem has a lot in common with job-shop scheduling. The main differences are (i) there are additional timing restrictions between production steps (e.g., there must be at most 4 hours between the end of the first production step and the start of the second production step for uni lacquers), and (ii) an order must use resources in parallel (every lacquer needs a mixing vessel during its production in parallel with other resources).

There are two additional features of the case study that need explanation. First, an *availability* factor is associated with every resource. This factor models the fraction of the time that a resource is available due to the working hours of the personnel. E.g., if a resource is operated by personnel that works in two 8 hour shifts from Monday 6 am to Friday 10 pm (i.e., 16×5 hours per week), then the availability factor of that resource equals $\frac{80}{168}$. This availability factor is used to approximate the working hours constraint: the production time of a resource is divided by the availability factor. Second, a *performance* factor is associated with every resource. This factor models the fraction of the time that a resource is unavailable due to break-downs or maintenance. The performance factor is used in the same way as the availability factor to extend the processing times.

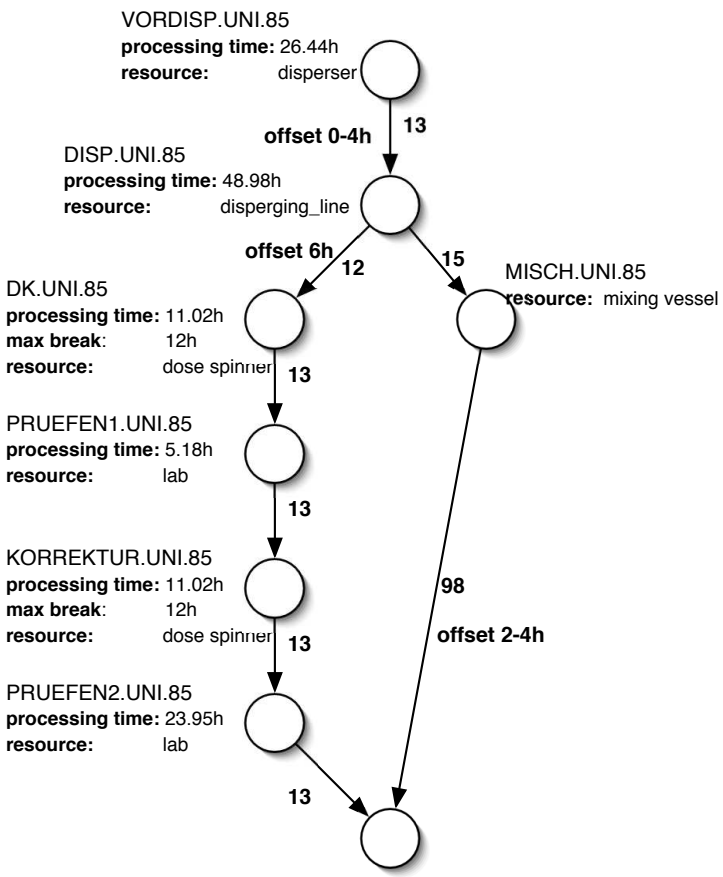

Figure 1: Recipe for uni lacquers as provided by AXXOM. Note that the edges are not necessarily precedences; the meaning of an edge is specified by the number that it has been labeled with (12, 13, 15 or 98).

Three different versions of the case study have been examined:

1. *Basic case study.* The performance factors are left out, but the availability factors are considered. Furthermore, the processing times do not depend on the size of an order (i.e., producing 15000 kg takes as much time as producing 5000 kg). Various instances (with 29 jobs, 72 jobs and 219 jobs) have been analyzed to check scalability of the approach.

2. *Extended case study.* The performance factors are now included, the processing times are a function of the size of an order, and storage and delay costs are added to quantify the feasibility of schedules. Furthermore, the model of some resources has been made more exact (for instance to model setup times between processing steps). There are two instances: a version with the availability factors and a version in which the availability factors have been replaced by an exact model of the working hours constraint.

3. *Stochastic case study.* This case concerns the performance factors and has been addressed in a separate paper ([5]), which is discussed in Section 6.

4 Modelling

This section explains how the problem has been modelled with timed automata. We intend to make the models that have been used in this paper available on the AMETIST website as soon as possible.

4.1 Information Transfer from Industry

A substantial amount of the time spent on the case study went into the modelling activities. The most difficult part here was the information transfer from AXXOM to the academic partners. In the first place, there was a language problem regarding the domain specific interpretation of terminology. For this purpose we compiled an initial dictionary in which relevant terms used are explained in natural language. This dictionary served as an agreement with AXXOM on the main, basic facts. Additionally, there was a documentation problem, regarding the (implicit) knowledge that always exists beyond any written specification. This problem remained even after agreeing on the dictionary. This suggests that, beyond a dictionary, additional validation of the basic facts would be desirable.

Another difficulty was caused by the format that AXXOM used for the recipes, which was neither standard, nor intuitive (see Figure 1). A better (from the computer science perspective, at least) representation had to be devised, resulting in Figure 2. This new notation also helped to detect other gaps in the case description.

Finally, AXXOM is not working on lacquer production, but develops tools for value chain management. The case description they provided is to some extent the description of their own model of the original case. Making our timed automata models we faced the problem that we were remodelling another model, tailored for another tool, rather than the original case. One example in point are the occurrence of very high delay costs. In the AXXOM tool they are needed to simulate hard deadlines by using (soft) due dates. In timed automata hard deadlines can be modelled directly.

4.2 Timed Automaton Models

The lacquer production case is very similar to the job shop scheduling problem, involving just a few additional timing constraints, and the basic modelling by timed automata roughly follows [2]. Each processing step can be mapped to a sequence of three locations in a timed automaton (fragment), see Figure 3, where the transition between the first two locations claims the resource, the second location represents the processing period, and the transition to the last location frees the resource.

The sequential and interleaved composition of the automaton fragments follows the descriptions and timing restrictions in the recipes. For each recipe there is a timed automaton (template) with free parameters for earliest start date and due date. Five resources are modelled as counters, and the remaining resources are modelled as small automata (since these resources need their own clock). There are altogether 29 (resp. 73 and 219) instantiations of the recipe automata with the

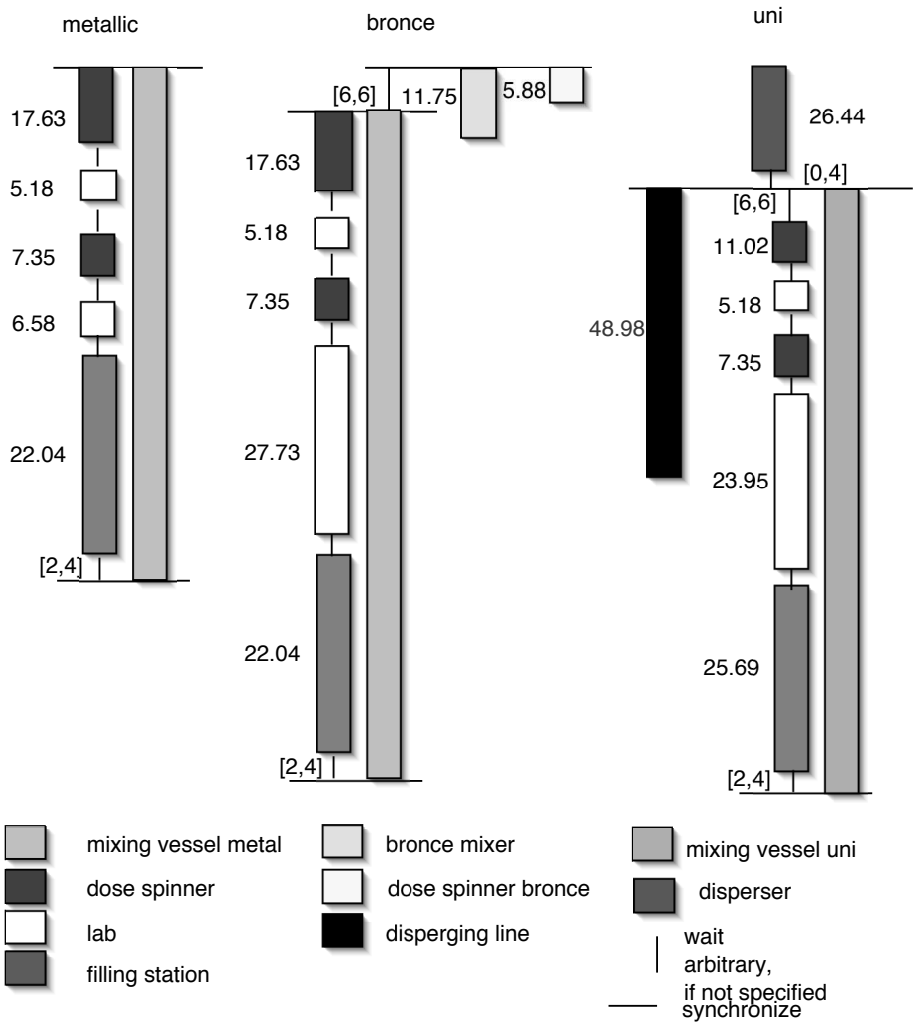

Figure 2: An alternative graphical representation for the three recipes.

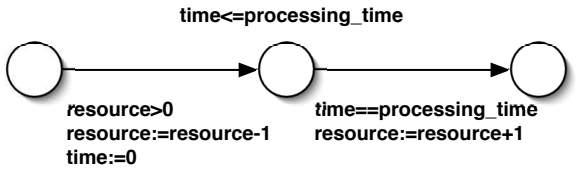

Figure 3: A single processing step modelled as timed automaton fragment.

example data for orders. The parallel composition of the instantiated automata and the resource automata forms the system model.

When looking for feasible schedules we checked the reachability property "all orders (automata representing an order) reach their final state", where a guard in the model only allowed to enter the final state if the due date has not passed already.

4.3 Modelling Heuristics

The heuristics we used are more or less standard in operations research, and are thus not specific for this case study. For instance, the "non-laziness" heuristic as explained below is the same as considering only "active" schedules [11]. The modelling of these heuristics can be seen as standard patterns that can be re-used for similar cases.

Each heuristic reduces the search space. We distinguish two kinds of heuristics. First, there are "nice" heuristics, for which we know that for each good schedule that was pruned away there is a schedule in the remaining search space that is at least as good. Second, there are "cut-and-pray" heuristics for which there is no such guarantee (i.e., the optimal schedule may be lost). Below we describe each of the heuristics we used, and show our modelling into the timed-automaton framework.

Non-overtaking

This heuristic is applied within each group of orders following the same recipe. It says, that an order started earlier also will get critical resources earlier than an order started later. This heuristics makes sense if for every two orders it holds that if the start time of the first order is smaller than the start time of the second order, then the end time of the first order is smaller than the end time of the second order. It is easy to see that for two orders following the same recipe, a non-overtaking schedule can be constructed from a schedule *with overtaking*. This can be done if at each moment when a resource is assigned to the later order (overtaking moment), we give it instead to the earlier order. This obviously is also a "nice" heuristic, if the time-spans have the same length.

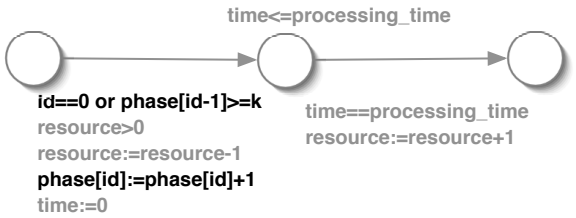

Figure 4: A timed automaton fragment for taking a resource with non-overtaking.

Technically, non-overtaking was realized by (indexed) counters *phase[id]*, one for each order having the number *id*. Note, that the sequence of identification numbers *id* reflects the sequence of starting times (and due dates, because the maximal production periods are the same). An restriction when taking a resource is, that the previous order *id-1* already has already taken the resource. When an order is taken, the counter *phase[id]* is increased. In Figure 4 we extended the basic timed-automaton fragment of Figure 3 by the counter construction; the original fragment is gray, the extensions are black. Note also that we have a set of counters for each of the three recipes (e.g., an order for metallic lacquer may overtake an order for a uni lacquer).

Non-laziness

In operations research non-lazy schedules are called active. The following behaviour is excluded: a process needs a resource that is available, but it does not take the resource. Instead, the resource *remains unused*, no other process takes it. Then, after a period of waiting the process decides to take the resource. (And we regard this waiting time as wasted, which is only true if there are no timing requirements for starting moments of subsequent processes.) This is a "nice" heuristic.

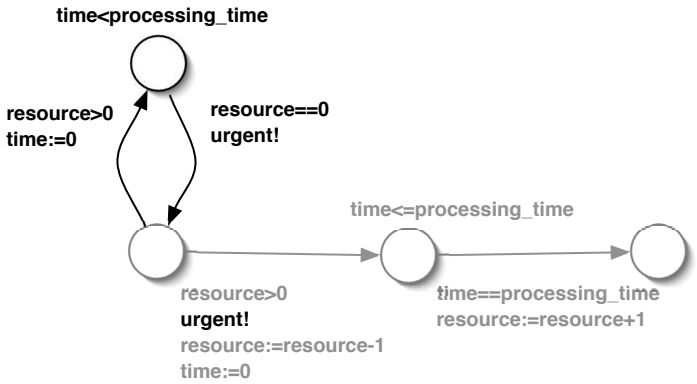

Figure 5: A timed automaton fragment for taking a resource with nonlaziness.

Technically, we extended the basic timed automaton fragment of Figure 3 by an extra location that is entered if the resource is available, but not taken as depicted in Figure 5; again, the original timed automaton fragment is gray, the additional construction for non-laziness is black. The new location can only be left, if there is another order taking the resource. If for *processing_time* the resource has not been taken, a deadlock is caused, which has the effect of backtracking and searching for other solutions. The intuition is, that if the resource has not been taken for *processing_time* the actual order could have taken it without being in the way for another order. Note that we use urgent communication on channel *urgent* so that some transitions are taken immediately if their guards become true, or pre-empted immediately by another enabled transition, if it exists. To make this work an automaton continuously offering synchronization on the *urgent* channel by a simple selfloop in its only location is part of the model. In the initial location of the automaton of Figure 5, therefore, when a resource becomes available it is either taken immediately or the idling state is reached immediately.

Greediness

This is a "cut-and-pray" heuristic. If there is a process step that needs a resource that is available, then the process step claims this resource immediately. By this it excludes possibly better schedules where some other (more important, because closer to deadline) process would claim the same resource shortly later. Note that greediness is stronger than non-laziness, i.e., every greedy schedule is non-lazy.

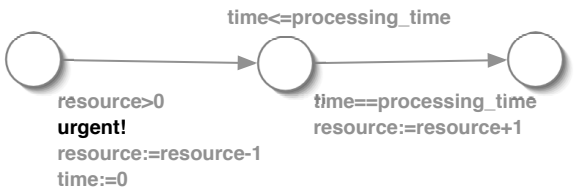

Figure 6: A timed automaton fragment for taking a resource with greediness.

The modelling of greediness in a timed automaton is easy: the requirement is that a resource has to be taken as soon as it is available. The communication via an urgent channel forces to take the transition as soon as the guard *resource>0*, is true, see Figure 6.

Reducing the Number of Active Orders

When not restricting the number of active orders (i.e. the orders that are processed at a certain moment), it often happens that many processes fight for the same resources, and block other resources while they wait. In our example the dose spinners (2 instances of these available) have to be used by each process twice, which makes them the most critical resource. Restricting the overall number of active orders avoids analysis of behaviour that is likely to be ineffective. This heuristic was very powerful, but belongs to the "cut-and-pray" type.

Technically, we realized this heuristics by a global variable that is increased when an order starts and decreased when an order is finished. A start condition for an order is that the counter has not reached its upper bound.

Increasing the Earliest Starting Time of Orders

This is a very simple heuristic that we use in the models that include costs. Ideally, an order is finished right on its deadline: it then has neither storage nor delay costs. Thus, when many orders are finished too early, their starting times may be increased to reduce the costs.

4.4 Modelling the Extended Case Study

Some constraints have been approximated in the basic case study to simplify the problem. In this section we discuss the extension of the model to cope with the full constraints. We begin with an informal explanation of these constraints.

First, there are setup times and costs. The filling lines must be cleaned between two consecutive orders if those orders are not of the same type. Thus, additional cleaning time (5 – 20 hours) is needed and there is a certain cost involved with cleaning. Modelling this constraint poses no problems. Instead of modelling the filling lines by an integer variable, they are now each modelled by an automaton that keeps track of the type of the order that has last been processed by it.

Second, there are delay and storage costs. The happiness of a customer decreases linearly with the lateness of his order. Thus, each order has a delay cost,

which is a "penalty" measured in euros per minute. Similarly, if an order is finished too early, then it has to be stored and this also costs a certain amount of euros per minute. In the initial problem, the costs are approximated by requiring that every order must be finished before its deadline. A more refined cost model enables us to prefer an order that is five minutes late above an order that is weeks early. UPPAAL CORA[1] is a version of UPPAAL for cost optimal reachability analysis in *linearly priced timed automata*. UPPAAL CORA enables us to model delay and storage costs in a natural way [7]. It allows the representation of costs as affine functions of the clock variables. For instance, Figure 7 depicts how delay costs are modeled. Every order has a delay cost factor (*dcf*) that gives the cost per timeunit when the order is too late. Furthermore, every job with id *id* has a bit *mlate[id]* that is 0 when the due date of the order has not yet passed, and 1 otherwise. Every location of the order automaton in which the order is not yet finished then is equipped with a specification of the time-derivative of the cost: *cost'==mlate[id]*dcf*. A similar strategy is followed for modelling the storage costs.

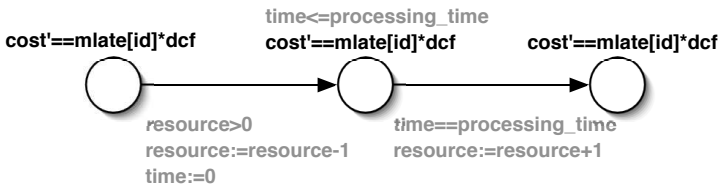

Figure 7: A timed automaton fragment for costs.

Third, there is the working hours constraint. The lacquer production is overseen by personnel that works in two or three shifts, depending on the machine they operate. Furthermore, the production is interrupted in weekends. Note that this constrained is approximated in the initial problem by the *availability factor* of machines. Another complicating factor is that some production steps may only be interrupted for 12 hours. Modelling the working hours constraint proved to be quite involved. A separate automaton was added that computes the *effective* processing time e, given the current time and the net processing time c. For instance, if the current time and c are such that the processing must be interrupted, then $e = c + B$, where B equals the length of the interruption. The additional automaton is rather big and laborious to produce, but quite logical in structure.

5 Model Checking Experiments

In Table 1 we collected models and model checking experiments for the feasibility analysis and schedule synthesis. The results were obtained using UPPAAL 3.4.6 on a laptop with an AMD Duron processor of 1GHz and with 512MB memory running under Linux Red Hat 9.0.

[1] `http://www.cs.aau.dk/~behrmann/cora/`

Table 1: Characteristics of models and experiments. Abbreviations in the "working hours" column (wh) are: -:no modelling, av:availability factors, and ex:explicit modeling. Abbreviations in the "heuristics" column are: g:greedy, nl:non-lazy, es:increment of earliest start, and no:non-overtaking. The "-" in the "time" column expresses that the search was stopped after 1 minute. All measurements, except the last one used a depth-first search order. The last model is comparable to AXXOM's model. The star means that we have taken 6000 random-best-depth-first runs each stopped after 3 seconds.

orders	**wh**	**costs**	**heuristics**	**act**	**time**	**min. cost found**
29	-	no	-	-	-	-
29	-	no	nl	-	1s	-
29	-	no	nl no	-	1s	-
29	-	no	g	-	<1s	-
29	-	no	g no	-	<1s	-
29	av	no	nl	-	2s	-
29	av	no	nl no	-	1s	-
29	av	no	g	-	<1s	-
29	av	no	g no	-	1s	-
73	-	no	-	-	-	-
73	-	no	nl	-	-	-
73	-	no	nl no	-	-	-
73	-	no	nl no	3	7s	-
73	-	no	nl no	4	27s	-
73	-	no	g	-	-	-
73	-	no	g no	-	53s	-
73	-	no	g no	4	7s	-
73	-	no	g no	3	5s	-
73	av	no	nl	-	-	-
73	av	no	nl no	-	-	-
73	av	no	nl no	3	-	-
73	av	no	nl no	4	3s	-
73	av	no	g	-	-	-
73	av	no	g no	-	-	-
73	av	no	g no	3	-	-
73	av	no	g no	4	3s	-
73	av	no	g no	5	5s	-
219	-	no	g no	4	8s	-
29	-	yes	es no	-	-	575,177
29	-	yes	es no nl	-	-	642,719
29	-	yes	es no g	-	-	714,204
29	av	yes	es no	-	-	1,145,404
29	av	yes	es no nl	-	-	802,643
29	av	yes	es no g	-	-	902,750
29	ex	yes	-	5	$\star$	3,149,075

The instance with 219 jobs reveled a number of scalability problems in the initial models and in UPPAAL. In particular, using 222 clocks results in symbolic states of approximately 200kB in size. The solution to this problem was twofold: First the models were changed to reuse clocks between jobs. In particular, the heuristic limiting the number of active jobs also provides a limit on the number of clocks needed; and the non-overtaking heuristic provides an easy way of uniquely assigning shared clocks to jobs since the starting order of jobs of a particular type is fixed. This change reduced the number of clocks to $3 \cdot A + 3$, where A is the maximum number of active jobs. Second, the 219 job experiment was made with a development version of UPPAAL. In particular, this version of UPPAAL optimizes the successor computation for models with many deadlocks and it fixes a bottleneck triggered by models with many processes using urgent channels. With these changes it is clear that our approach scales quite well.

The results show that non-laziness is sufficient for the case of 29 jobs. When using the availability factors, non-overtaking might have a small positive effect. This changes when we go to 73 orders. Greediness as the more aggressive heuristic gives no (fast) results, but together with non-overtaking the search terminates within a minute. With additional restriction of the active orders the results come much faster. For non-laziness we get only (fast) results if non-overtaking and restriction of the active jobs is added. The experiments also show that the good upper bound for the number of active jobs can vary in different settings and can only determined during experimentation.

Experiments have been performed also for the extended version of the case study using UPPAAL CORA. For the models without the exact modelling of the working hours constraint, it seems that using non-laziness is better than greediness. Furthermore, it is essential to increase the earliest starting time of orders (otherwise they are finished much too early resulting in large storage costs). A schedule with cost 575,177 is equivalent to the situation in which every order is finished 35 hours too early (or every order is finished 50 minutes too late). Due to the enormous size of the state space, however, we are not able to tell whether this is the lowest possible cost. Introducing an exact model for the working hours makes the problem significantly more complex since the symbolic state space is severely fragmented and more deadlocks are present. Still, we derived schedules that are competitive with those provided by AXXOM.

It should be noted that the non-laziness heuristic is not applicable to the extended case, since storage costs make it profitable to be lazy. Also the non-overtaking heuristic is rendered a "cut-and-pray" by the addition of costs. The greediness heuristic is not applicable either, but this is due to a limitation in UPPAAL. We currently work on adapting all these heuristics to the extended case and expect the experiments of the final version of this paper to be based on those new models. These models will also incorporate the technique used in the basic case study for reducing the number of clocks.

6 Stochastic Analysis

As explained earlier, so-called performance factors are used to indicate the percentage of time that a resource is unavailable due to maintenance and break-downs. The way in which Axxom deals with this information is that the processing time on each resource is extended by the corresponding factor. E.g., if a machine only is available half of the time, the processing time for each processing step using this resource is doubled. Schedules are derived assuming that the process durations are extended in this way. This raised the question on the interpretation of the schedules derived with the extended processing times. Stochastic analysis [5] showed that the schedules derived in this way have less chance to reach the due dates than schedules without extended times. The interpretation roughly is as follows: if we reserve time for break-down when a resource is actually available, this time is simply wasted. Later, when the resource really breaks down, there will be too little time left to reach the due date. A conclusion is that extending processing times may give a useful indication how many orders can probably be done within a long time interval, say a few months, but it does not help for daily fine-tuned scheduling.

7 Evaluation and Conclusion

We have shown that feasible schedules for a lacquer production case can be derived doing real-time reachability analysis with the timed automata model checker Uppaal. We could treat instances with 29 orders within 1 second, and the extensions to 73 and 219 orders did not significantly increase the computation times (all less than 10 seconds, given the right heuristics). To deal with the full set of constraints of the original problem we had to introduce costs into the model, viz., setup costs for filling stations, storage costs for orders that are produced too early, and delay costs for orders that are too late. This transformed the problem into a cost-optimization problem, which was treated using Uppaal Cora, a cost-optimizing version of Uppaal.

A further extension of the model was needed to deal with the so-called working hours constraints, which increased the size and complexity of the model significantly. Yet, also for this case competitive schedules could be derived using Uppaal Cora.

On the one hand, it is clear that this application of model checking techniques to this kind of production scheduling problems is not (yet) push-button technology: yo obtain results our models had to be constructed with care, and the right heuristics had to be identified. On the other hand, it is reasonable to assume that many production scheduling problems have similar ingredients and that modelling techniques and patterns for typical plant processes and heuristics can be reused. Further experiments have to be carried out to identify a useful core collection of such modelling patterns.

Of course, there still are a number of open issues. One important question is the extent to which our approach scales up. We treated 29 orders in the first experiments, and 72 and 219 in subsequent ones, which did not require significantly

more time. We believe that this is due to the heuristic that limits the number of active orders, which also limits the complexity that we can handle in larger problems. Currently, we are working on a case involving some two thousand orders. One of the problems here is even to construct a representative case of this size.

The use of performance and availability factors leads to questions of interpretation. Extending the processing times by these factors can be used to analyze how many orders should be feasible on a longer time scale. The stochastic analysis in [5], however, has shown that using performance and availability factors to obtain concrete schedules increases the probability to miss deadlines. The use of these factors thus makes models inherently approximative, and it does not seem very useful to include finer information about penalties (such as setup and cleaning costs) into the model, as is the case now. It is unclear what modelling assumptions are best suitable for the derivation of concrete short-term schedules, where storage costs have to be minimized and delay costs to be avoided. An idea that we want to explore is that of using a form of schedule refinement taking rough long-term schedules as a basis for obtaining precise schedules for concrete short-term. A transformation approach to scheduling, although in a different context, was successfully used in another case study of the AMETIST project, viz., the CYBERNETIX case [9]. Another idea that will be investigated is that of searching for schedules in reverse time, starting from the due dates of orders; valid schedules obtained this way avoid storage and delay costs by construction.

The case study also raised a number of pragmatic questions concerning the modelling process. It turned out to be nontrivial to obtain all relevant information from AXXOM. In spite of our efforts to create a dictionary and better graphical representations, the models had to be changed substantially in an advanced stage of the project, as initially provided requirements turned out to be over-specified. The experience suggests that beyond a dictionary, there should have been some joint activity to certify the informal explanations. A related aspect is that the problem description of AXXOM was strongly influenced by the capabilities of their own planning tool. This implies that in some places we may have been remodelling the AXXOM model, rather than modelling the original problem.

Summarizing, we can say that our experience with the AXXOM case study shows that the application of model checking techniques for production scheduling is very promising. Still, considerable further work on modelling methods, reusability of modelling patterns, identification and evaluation of heuristics – all in the context of case studies of greater orders of magnitude – is needed to develop it into a readily applicable standard technique for schedule synthesis.

Acknowledgements

We would like to acknowledge the essential role played by Dagmar Ludewig and Sonja Loeschmann of AXXOM, who were always willing to provide the answers to our many questions concerning this case study.

References

[1] Y. Abdeddaim, A. Kerba, and O. Maler. Task graph scheduling using timed automata. In *8th International Workshop on Formal Methods for Parallel Programming: Theory and Applications (FMPPTA'03)*, 2003.

[2] Y. Abdeddaim and O Maler. Job-shop scheduling using timed automata. In G. Berry, H. Comon, and A. Finkel, editors, *CAV 2001*, number 2102 in LNCS, pages 478–492. Springer–Verlag, 2001.

[3] R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(138):183–235, 1994.

[4] European Community Project IST-2001-35304 AMETIST (Advanced Methods for Timed Systems). `http://ametist.cs.utwente.nl/`.

[5] H. C. Bohnenkamp, H. Hermanns, R. Klaren, A. Mader, and Y. S. Usenko. Synthesis and stochastic assessment of schedules for lacquer production. In *1st Int. Conf. on Quantitative Evaluation of Systems (QEST)*, pages 28–37, Enschede, The Netherlands, Sep 2004. IEEE Computer Society Press, Los Alamitos, California.

[6] A. Fehnker. Scheduling a Steel Plant with Timed Automata. In *Sixth International Conference on Real-Time Computing Systems and Applications (RTCSA'99)*, pages 280–287. IEEE Computer Society Press, 1999.

[7] K. G. Larsen, G. Behrmann, E. Brinksma, A. Fehnker, T. Hune, P. Pettersson, and J. Romijn. As cheap as possible: Efficient cost-optimal reachability for priced timed automata. In G. Berry, H. Comon, and A. Finkel, editors, *CAV 2001*, number 2102 in LNCS, pages 493–505. Springer–Verlag, 2001.

[8] K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1/2):134–152, 1997.

[9] A. Mader. Deriving schedules for the CYBERNETIX case study, 2003.

[10] O. Maler, A. Pnueli, and J. Sifakis. On the synthesis of discrete controllers for timed systems. In *Proceedings of STACS'95*, volume 900 of *LNCS*. Springer-Verlag, 1995.

[11] M. Pinedo. *Scheduling: Theory, Algorithms and Systems.* Prentice Hall, 2002.

[12] M. Pinedo and X. Chao. *Operations Scheduling with Applications in Manufacturing Systems.* McGraw-Hill, 1999.

[13] S. Yovine. KRONOS: a verification tool for real-time systems. *International Journal on Software Tools for Technology Transfer*, 1(1/2):123–133, 1997.